HICOLOR - 256 and 4096 Color Modes

Table of Contents

[Overview 2](#_Toc411668140)

[Timing 3](#_Toc411668141)

[SdramCommands 3](#_Toc411668142)

[VRAM Layouts 3](#_Toc411668143)

[VRAM Layout within the SDRAM chip 4](#_Toc411668144)

# Overview

The two hi-color modes require significant VRAM in order to store their images. More VRAM than the Z80 address space. More VRAM than the FPGA has internally. Luckily, there is a very large SDRAM chip attached to the FPGA. We'll use this, but we'll need to deal with refreshing. The only big drawback is that the CPU will take a long time to load data into this memory.

The 256 Color mode uses only the low-byte of each memory address, (one address per-pixel), and performs a 256-color Palette Lookup using that byte, before storing the pixel into the scanline buffer. Index 0 is considered to be transparent. The hi-color layer has a global depth register value of 0..3, just like the tiled backgrounds. The BG pixel is generated at the same time as the HICOLOR pixel, so as to determine who wins by priority at the very moment that the background data is to be written to the Scanline Buffer.

The 4096 Color mode uses 2 bytes per pixel, and performs no palette lookup. Instead, it performs a 16-bit SDRAM read to fetch the 12 bits of RGB data, as well as 2 bits of optional per-pixel depth, and a transparency bit. The top 4 bits have the following meanings:

* 1 bit of transparency. This feature is optional, and can be disabled using a video register. 1 = transparent, 0 = opaque.
* 1 bit unused.
* 2 bits of priority. Thus, the RGB image can have per-pixel priority, and stitch into the final image. This feature is optional, and can be overridden to be a constant priority, using video registers. Regarding a tie, the HICOLOR image wins. Thus, the tendency is that the image lie on top of the final background (as-if it were a HUD).

The SDRAM Controller is guaranteed to visit every scanline of SDRAM on every frame, even if the layer is hidden, or if scanlines are re-arranged by the copper. This ensures that the SDRAM is refreshed automatically, since the bank and row are Activated (opened), and then Precharged (closed) at regular intervals. SDRAM outside of the display area is not refreshed, but is also not of any value.

The CPU communicates entirely through a few Video Registers:

* XXX XXXXXXXX POSITION X,Y: Pixel to begin reading/writing.  
  YY YYYYYYYY NOTE: Writing this, causes the hardware to queue an SDRAM READ of the  
   new address. Writes to this while BUSY will work, but will result in an  
   incorrect VALUE when you eventually read it.  
   (BUSY will go true until VALUE is available for reading).
* VVVVVVVV VALUE V=0..255.  
   READ: Fetch byte from VRAM.   
   WRITE: Store byte into VRAM.  
   Note that reads and writes must be made only when not BUSY.  
   Reading or writing causes the internal address to auto-increment   
   within the scanline.  
   It will auto-advance to the next scanline.

Reading or writing while BUSY, will yield unpredictable results.

* VTCLLPDD CONTROL V:0=layer is hidden, 1= layer is visible  
   T: 0=No Transparency, 1=Transparency (index 0 or RGB 000)  
   C:0=256 colors (1 bpp), 1=4096 colors (2 bpp)  
   LL: Layout 0= 4x1 (2048x256)  
   1=2x2 (1024x512)  
   2=1x4 (512x1024)  
   P: Per-Pixel depth (0=off, 1=on if 4096 mode.   
   Probably works for 256-color mode too)  
   D: Layer Depth (0..3)
* xxxxxXXX XXXXXXXX HSCROLL X=0..2047 (horizontal scroll value)
* xxxxxxYY YYYYYYYY VSCROLL Y=0..1023 (vertical scroll value)
* xxxxxxxB BUSY B: Busy (ReadOnly)  
   0=Byte can be read or written.  
   1=Not ready for read, write, or new scanline.   
   Check again later.

# HiColor Register State Machine

The BUSY flag

# Timing

The Video Clock is running at 62Mhz.

There are a total of 3360 clocks per virtual scanline (two physical scanlines).

It will cost about 650 cycles to pull 640 bytes of data from SDRAM

62Mhz = 16.1290ns

200,000ns = 12401 clocks

60ns = 4 clocks

42ns = 3 clocks

15ns = 1 clock

64ms = 4.6 frames @72Hz (thus, all of SDRAM must be refreshed within about 4 frames).

# Sdram State Machine

The state machine allows a number of transactions to occur (sequentially).

* Read one byte from SDRAM, specified by the CPU.
* Write one byte to SDRAM, specified by the CPU.
* Stream 1 scanline's worth of data from SDRAM to the REnderer (320 addresses).
* Perform a refresh of the next row of SDRAM (sequentially).

# VRAM Layouts

The SDRAM can access up to 4 banks of 512 x 16-bit values at once.

I use all 16 bits for 4096-color mode.

I use only the bottom 8 bits in 256-color mode.

Thus, all of the pixel address generation code remains the same, regardless of video mode.

The CPU byte read/write code does change, depending upon video mode. In 4096-color mode, it requires two byte writes per address increment. It toggles UDQM and LDQM. In 256-color mode, it uses only LDQM, and increments the address on every byte write.

There are 256 ROWS. One row is activated for each scanline. They will be in sequence, unless the COPPER re-arranges the scanlines.

Because of this, the scanline rendering cannot be relied upon to open every row to keep everything refreshed. Instead, immediately after the scanline render completes and the current rows are closed, REFRESH rows are opened (i.e. based on a refresh counter). This ensures that all rows are refreshed within the necessary period. The refresh counter simply increments on every scanline render, and ignores the scanline and VSCROLL values. It costs 8 clocks to activate (open) all 4 banks of the current REFRESH ROW count, 1 clock to delay 15ns after the last activate, 1 clock to precharge (close) all banks, and 1 final clock to delay 15ns after the precharge. That's a total of 7 clocks. Not bad, considering there are about 3360 clocks per scanline. All of the remaining time is available to the CPU for reading and writing data in VRAM. Because CPU reads and writes may be blocked by the scanline rendering or refresh activity, there is a BUSY flag in a video register, indicating whether the last CPU HICOLOR read/write action has completed yet.

Every SDRAM row must be visited every 64ms. I visit 250 scanlines every frame. At 72Hz, I can visit 250 rows in 13.8889ms. That means, I can refresh 4.6x the height of the screen before I reach 64ms and must start over. Lets round that back to 4.096x250 scanlines = 1024 scanlines (rows) of SDRAM. By opening all 4 banks, I get 512 x 4 = 2048 pixels wide. Thus, maximum HICOLOR screen dimensions are 2048x1024 pixels (6.4 x 4.2 screens in size).

The HICOLOR rendering happens at the same time as BG rendering, and the results are merged on the fly.

# VRAM Layout within the SDRAM chip

There are 3 layouts:

0: Wide (2048 x 256)  
1: Box (1024 x 512)  
2: Tall (512 x 1024)  
3: (unused)

All 4 banks get opened at the beginning of each scanline, as follows:

ROW = (VSCROLL + scanline)[7:0]

BANK = (layout == BOX) ? X[10:9] xor Y[8:9] : X[10:9] xor Y[9:8];

Thus, the following 3 bank layouts are possible:

Wide: 0123

Box: 01  
 23

Tall: 0  
 1  
 2  
 3

Wrap masks:

HWRAPMASK = (layout == WIDE) ? 2047 :  
 (layout == BOX) ? 1023 :  
 511; // TALL

VWRAPMASK = (layout == WIDE) ? 255 :  
 (layout == BOX) ? 511 :  
 1023; // TALL

Each pixel is looked-up as follows:

COLUMN = HSCROLL[8:0] & HWRAPMASK

How the data within them is accessed, depends upon the Layout configuration bits.

# Filled Polygons

All polys are 3-point (ABC).

A is the highest point on-screen.  
B is the lowest point on-screen.  
C is a point somewhere in-between.

AB -> AC (ends when AC ends)  
AB -> CB (ends when CB ends)